1　A3C の擬似コード

Algorithm 1 各スレッド毎の A3C の擬似コード

Input:

//グローバルパラメータ θ と θ_v，グローバルカウンター $T = 0$ は全スレッドで共有する
// パラメータ θ' と θ'_v は各スレッド毎に保有する
$t \leftarrow 1$
while $T < T_{\max}$ **do**
　$d\theta \leftarrow 0,\ d\theta_v \leftarrow 0$
　$\theta' = \theta,\ \theta'_v = \theta_v$
　$t_{\text{start}} = t$
　$s_t,\ done_t$ を観測
　$done = done_t$
　while !$done$ and $t - t_{\text{start}} < t_{\max}$ **do**
　　$\pi(a_t|s_t;\theta')$ に従って a_t を実行
　　$(s_{t+1},\ r_t,\ done_t)$ を観測
　　$done = done_t$
　　$t \leftarrow t + 1$
　　$T \leftarrow T + 1$
　end while
　if $done$ **then**
　　$R = 0$
　else
　　$R = V(s_t,\ \theta'_v)$　// 最終状態からのブートストラップ
　end if
　for $i \in \{t-1,\ \cdots,\ t_{\text{start}}\}$ **do**
　　$R \leftarrow r_i + \gamma R$
　　$d\theta \leftarrow d\theta + \Delta_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$　// θ' に関する勾配を蓄積
　　$d\theta_v \leftarrow d\theta_v + \delta(R - V(s_i;\theta'_v))^2/\delta\theta'_v$　// θ'_v に関する勾配を蓄積
　end for
　$d\theta,\ d\theta_v$ を使って $\theta,\ \theta_v$ を非同期更新
end while

Output:

2　Atari Games の環境のラップ処理

Deep Q Network では Atari Games から受け取った状態の前処理などを行うことで学習をしやすくしている．以下にそれらの処理を示す．なお，$reset_{\text{hoge}}()$ は環境をリセットするときに呼ぶメソッドを，$step_{\text{hoge}}(a_t)$ は行動 a_t を取って環境を更新するときに呼ぶメソッドを，$observe_{\text{hoge}}()$ は環境の状態を返すときに呼ばれるメソッドを，$reward_{\text{hoge}}()$ は報酬を返すときに呼ばれるメソッドを表す．

Algorithm 2 $reset_{\text{noop}}()$

Input: env, l_{nomax}

エピソードの開始時に，数フレーム何もしない行動を取り，初期状態を決定する．

$T \sim U(1, l_{\text{nomax}})$

for $t' = 1, \cdots, T$ **do**

    $a_{t'}$ = (do nothing) の実行

end for

Output: 初期状態が決定した環境 env

Algorithm 3 $step_{\text{repeat}}(a_t)$

Input: $env, l_{\text{repeat}}, a_t$

1 回行動を取ると，同じ行動を指定フレーム続ける．指定数分行動を繰り返したら，直前のフレームの観測との最大値を状態として返す．

※ a_t は選択したい行動とする．

$r_{\text{total}} = 0$

for $t' = 1, \cdots, l_{\text{repeat}}$ **do**

    $s_{\text{prev}} = s_{t'}$

    $a_{t'} = a_t$ として行動を選択し，環境 env を更新，$(s_{t'+1}, r_{t'}, done_{t'})$ を観測する．

    $r_{\text{total}} = r_{\text{total}} + r_{t'}$

    $s_{\text{max}} = \max(s_{\text{prev}}, s_{t'+1})$

    $done = done_{t'}$

end for

Output: $s_{\text{max}}, r_{\text{total}}, done$

Algorithm 4 $observe_{\text{gray84}}()$

Input: s_t

観測した画面を (84,84) サイズのグレースケール画像に変換して返す．

s_t をグレースケール画像に変換

変換後の s_t をさらに (84,84) に reshape

Output: 変換後の s_t

Algorithm 5 $step_{\text{stack}}(a_t)$

Input: $env, l_{\text{history}}, a_t, S$

l_{history} ステップ数分の観測の履歴を状態として返す．

S は観測の履歴とする．

a_t を行動として選択し，環境 env を更新，$(s_{t+1}, r_t, done_t)$ を観測する．

S に s_{t+1} を追加．$|S| > l_{\text{history}}$ なら，一番古い履歴を S から削除する．

Output: $S, r_t, done_t$

Algorithm 6 $reward_{\text{clip}}()$

Input: r_t

報酬 r_t が正なら +1 に，負なら −1 に，0 なら 0 として返す．

if $r_t > 0$ **then**

  $r_t = 1$

else if $r_t < 0$ **then**

  $r_t = -1$

else

  $r_t = 0$

end if

Output: r_t